NAME

racluster.conf − **racluster** flow model definitions.

COPYRIGHT

SYNOPSIS

racluster.conf

DESCRIPTION

Programs that perform flexible aggregation of argus data, such as racluster(1) and radium(8), can be configured to aggregate using arbitrary flow models. This configuration file provides a syntax for flow matching and aggregation model assignments on a per flow basis, allowing for highly flexible aggregation strategies on a single argus stream.

The configuration file is structured as a set of initialization variables, and then followed by a collection of filter matching statements and model definitions. The concept is that one identifies specific Argus Flow Activity Records through specification of a Argus filter, and if the filter matches, then that record will be modified using the optional racluster aggregation model, and then aggregated with other records that matched the filter. If the filter does not match, racluster will "fall through" to the next filter.

OPTIONS

The aggregation clients have a small number of options for controlling specific aspects of aggregation function and output.

RACLUSTER_MODEL_NAME

Racluster configurations can be named. This is important for ra* aggregation programs that support multiple concurrent models at a time, so you can tell them apart. This is completely optional.

RACLUSTER_REPORT_AGGREGATION

Racluster, when it merges argus records together, adds a new aggregation metric to the resulting record, which reports on the number of records that were merged together and provides some additional statistical values that provide record arrival rates and mean record durations. By setting this option to "no", you can have racluster() not provide this metric. This is very important if you have multiple aggregatation processes 'piped' in order to generate complex aggregation reports. It is also useful when creating full-duplex records from half-duplex merging operations.

RACLUSTER_REPORT_AGGREGATION=yes

RACLUSTER_PRESERVE_FIELDS

All aggregation clients detect when a flow descriptor object would need to be modified to conform to an aggregation result. If you merge two records that have different source addresses, the merge process will modify the source addresses. For all fields there are specific algorithms used to perserving as much data as possible so that the result is 'PRESERVE'ing. For IP addresses, the algorithm preserves the longest prefix match. This is valuable information when attempting to discover trends.

However, some applications may want the resulting output to completely conform to the new flow definitions. In other words, if you are aggregatating and the source address is not in the key, the algorithm will zero out the soruce address prior to record merging.

To force racluster() like programs to convert flow descriptions to the new flow model descriptors, set this option to "no".

RACLUSTER_PRESERVE_FIELDS=yes

RACLUSTER_AUTO_CORRECTION

When aggregating Argus records together, all aggregation clients have the ability to autocorrect the assignment of flow initiator and receiver. This is important for processing Argus records derived from Cisco Netflow style flow monitors, and for Argus records that were generated by long lived flows that have extended idle periods. Because it is possible for ra* aggregation clients to receive half-duplex flow records, or multiple flow records for the same long live flow, autocorrecting the argus records allows the client aggregation process to match A -> B and B -> A records that belong to the same flow.

However, with certain flow aggregation models, the autocorrection logic can cause aggregation mis-match, merging dissimilar flow together. This is especially the case for complex aggregations where fields are ignored only on specific record types. As a result, when providing custom aggregation models, autocorrection is disabled by default.

If you would like to re-enable the autocorrection function, set this variable to "yes";

RACLUSTER_AUTO_CORRECTION=no

AGGREGATION CONFIGURATION

An aggregation configuration is composed of one or more aggregration statements. An aggregation statement contains, on a single line, a mandatory filter specification, and optionally a model specification, a status timer and/or an idle timer value. Specifications are formated as:

keyword'='["]value["]

Valid keywords are:
filter, model, status and idle.

Filter specifications are standard ra* client filters. See ra.1 for a complete description.

Model specifications are identical to the -m option support provided by racluster() on the command line.

Status and Idle timer values are specified in seconds and are usually not quoted, although they can be.

Argus record flow descriptors are compared to the filter that matches statements in sequential, or "fall through", order, much like existing Access Control List definitions supported by routers, switches and firewalls. These filters are

The model statement is used to modify the flow description of matching Argus records. Records are aggregated based on the modified flow descriptor that results from applying the flow model, and are only merged with other records that also matched this filter, which should eliminate problems with a independent

In each flow descriptor matching statement is a TimeOut period, which is how long the aggregator will hold the flow cache before reporting it, and an IdleTimeOut period, which is how long the aggregation process will hold the flow in its cache, if there is no activity.

If a record doesn't match any statement in the configuration, then it is aggregated based on its unmodified flow descriptor. This aggregates flow reports from the same long lived flow.

EXAMPLE

This configuration is not intended to do anything useful, at least, it is designed for demonstration purposes. With that said, lets get to it.

filter="icmp"
filter="arp" model="proto daddr"
filter="tcp or udp and dst port lt 1025" model="saddr daddr proto dport" status=120
filter="tcp or udp" model="saddr daddr proto sport dport" status=120 idle=3600
filter="" model="saddr daddr proto" status=0 idle=3600

All records are first tested as 'icmp' flows, then 'arp' and then 'tcp or udp' flows. If the records do not match these filters, they will match the 4th aggregation statement, as it has a null filter, which matches everything.

If a record matches the 1st statement, it is aggregated using the default aggregation model, which is the conventional 5 tuple flow model. For icmp, this includes the src and destination addresses, the proto field and the icmp type and code values. Because there is no status or idle timer values specified, the aggregator will merge records until EOF is encountered on the input stream, and output the single record and its stats at the end of the run.

If a record matches the 2nd statement, i.e. it is an arp flow, then the record flow descriptor is modified so that the protocol (arp) and the requestors source address are the only fields preserved. This would merge all the arp records requesting a specific IP addresss, and is useful for reducing arp traffic in the output stream or file. The idea is that you are interested in the response, not who made the request.

If a record matches the 3rd statement, i.e. it is a "tcp or udp" flow with the destination port less than 1025, the flow model ignores the soure port value, and merges records that match. The aggregator will generate a 120 second status report on the resulting flow, so that when there is activity, the aggregator will generate output that doesn't exceed 120 seconds in duration. If the flow is idle for 3600 seconds, the aggregator will

This is a very popular service oriented aggregation strategy, as it preserves the client address and the server address, along with the service port number, when its less than 1025, which are the Well Known Ports, or Reserved port number space.

If the flow is "udp or tcp" and the dst port is not less than 1025, then the flow will match the 4th statement, and the aggregator will use the default flow model, generating status records and timeing out.

The 5th statement, the "catch all" statement, specifies that the aggregator should preserve only the source and destination addresses, and the protcol number. No status records will be generated, but if it times out it will flush the record out.

This configuration strategy should provide support for any type of aggregation methodology you can dream up, well at least most that you will want to use.

SEE ALSO

racluster(1)